1.114.1
Perform security administration tasks
Weight 4

Linux Professional Institute Certification — 102

Nick Urbanik <nicku@nicku.org>

2005 November

Outline

- Context
- Objectives
- Configuring TCP Wrappers
  - Rules for `hosts.allow`, `hosts.deny`
  - Format of rules in `hosts.{allow,deny}`
  - Example
  - Wildcards
  - Patterns
  - Option Fields: Logging
  - `EXCEPT` operator
  - How is TCP Wrappers enabled?
- Finding files with SUID/SGID bit set
  - Effect of SUID/SGID permissions
  - Specifying Permissions to `find`
- Verify Packages
  - Why Verify Software Packages?
  - Verify Package Files with `rpm`
  - Verify Installed Packages with `rpm`
  - Verify Packages with `apt/dpkg`
- Passwords and Aging Information
- Update binaries with security alerts
- Basic intro to `iptables` and `ipchains`
  - `iptables` tables
  - `iptables` chains
  - Examples of filtering
  - Viewing firewall rules
  - Saving and restoring rules
- References

Topic 114 Security [8]

Where we are up to

1.114.1 **Perform security administration tasks [4]**

1.114.2 Setup host security [3]

1.114.3 Setup user level security [1]

Description of Objective

1.114.1 Perform security administration tasks [4]

Candidates should know how to review system configuration to ensure host security in accordance with local security policies. This objective includes how to configure TCP wrappers, find files with SUID/SGID bit set, verify packages, set or change user passwords and password aging information, update binaries as recommended by CERT, BUGTRAQ, and/or distribution's security alerts. Includes basic knowledge of ipchains and iptables.

Key files, terms, and utilities include:

1.114.1 Perform security administration tasks [4]

`/proc/net/ip_fwchains` — Firewall chain linkage (2.2 kernel)

`/proc/net/ip_fwnames` — Firewall chain names (2.2 kernel)

`/proc/net/ip_masquerade` — Major masquerading table (2.2 kernel)

`find` — We see how to find SUID and SGID programs using `find`

`ipchains` — The tool to configure the firewall on a 2.2 kernel

`passwd` — Discuss how to use to set password aging information

socket — The end point of a network connection

`iptables` — The tool to configure the firewall on a 2.4, 2.6 kernel

Rules for `hosts.allow`, `hosts.deny`

- Search stops at the first match *in this sequence*:
- Access will be granted when a (daemon,client) pair matches an entry in the `/etc/hosts.allow` file.
- Otherwise, access will be denied when a (daemon,client) pair matches an entry in the `/etc/hosts.deny` file.
- Otherwise, access will be granted.

More about how tcp_wrappers rules applied

- Because access rules in `hosts.allow` are applied first, they take precedence over rules specified in `hosts.deny`.
  - Therefore, if access to a service is allowed in `hosts.allow`, a rule denying access to that same service in `hosts.deny` is ignored.
- The rules in each file are read from the top down and the first matching rule for a given service is the only one applied. The *order of the rules is extremely important*.
- If no rules for the service are found in either file, or if neither file exists, access to the service is *granted*.
- changes to `hosts.allow` or `hosts.deny` *take effect immediately* without restarting network services.

Format of rules in `hosts.{allow,deny}`

- Each rule is of the form:
  ⟨*daemon list*⟩ `:` ⟨*client list*⟩ `[:` ⟨*option*⟩ `:` ⟨*option*⟩ `: ...]`

⟨*daemon list*⟩ A comma separated list of process names (*not service names*) or the `ALL` wildcard — see § **??**. The daemon list also accepts the `EXCEPT` operator to allow greater flexibility — see § **??**

⟨*client list*⟩ A comma separated list of hostnames, host IP addresses, special patterns — see § **??**, or special wildcards — see § **??** — which identify the hosts effected by the rule. You can also use the `EXCEPT` operator.

⟨*option*⟩ An optional action or colon separated list of actions performed when the rule is triggered. Option fields support % expansions — see `$ man 5 hosts_access` ↩, launch shell commands, allow or deny access, and *alter logging behavior* — see § **??**

Example rule

```
vsftpd :  .example.com
```

- watch for connections to the FTP daemon (`vsftpd`) from any host in the `example.com` domain.
- If this rule appears in `hosts.allow`, the connection is accepted.
- If this rule appears in `hosts.deny`, the connection is rejected.

Wildcards

Wildcards allow TCP wrappers to more easily match groups of daemons or hosts. They are used most frequently in the client list field of access rules.

The following wildcards may be used:

`ALL` Matches everything. It can be used for both the daemon list and the client list.

`LOCAL` Matches any host that does not contain a period (.), such as `localhost`

`KNOWN` Matches any IP address which has a corresponding hostname; also matches usernames when the *ident* service is available (which is usually not)

`UNKNOWN` Matches any IP address which does *not* have a corresponding hostname; also matches usernames when the *ident* service *not* available

`PARANOID` Matches any host where a double reverse hostname/IP address lookup fails to match

Patterns

- Hostname beginning with a period (`.`) This matches any host in the `example.com` domain:
  `ALL : .example.com`
- IP address ending with a period (.) This matches any host in the 192.168.x.x network:
  `ALL : 192.168.`
- IP address/netmask pair This matches any host in the address range 192.168.0.0 . . . 192.168.1.255:
  `ALL : 192.168.0.0/255.255.254.0`
  - Note: a pattern of the form `192.168.0.0/23` *will not work*
- The asterisk (*) Asterisks can match entire groups of hostnames or IP addresses. This matches any host in the `example.com` domain:
  `ALL : *.example.com`
  - This asterisk notation is not used anywhere else as far as I know.

Context

Objectives

Configuring TCP Wrappers

Rules for hosts.allow, hosts.deny

Format of rules in hosts.{allow,deny}

Example

Wildcards

Patterns

Option Fields: Logging

EXCEPT operator

How is TCP Wrappers enabled?

Finding files with SUID/SGID bit set

Verify Packages

Passwords and Aging Information

Update binaries with security alerts

Basic intro to iptables and ipchains

References

License Of This Document

Option Fields: Logging with severity

- See $ **man 5 hosts_options** ↩ for details of other options; just look at severity directive for logging access
- Here, connections to the SSH daemon from any host in the example.com domain are logged to the default authpriv syslog facility (because no facility value is specified) with a level of emerg:
  `sshd : .example.com : severity emerg`
- specifying a facility: The following example logs any SSH connection attempts by hosts from the example.com domain to the local0 facility with a level of alert:
  `sshd : .example.com : severity local0.alert`

`EXCEPT` operator

- There is one operator: `EXCEPT`.
- can be used in both the daemon list and the client list of a rule.
- allows specific exceptions to broader matches within the same rule.
- Example:

  ```
  ALL: .example.com EXCEPT
  cracker.example.com
  ```
- In the another example from a `hosts.allow` file, clients from the 192.168.0.*x* network can use all services except for FTP:

  ```
  ALL EXCEPT vsftpd:  192.168.0.
  ```

How is TCP Wrappers enabled?

- Recent systems use `libwrap`, part of the `tcp_wrappers` package
- Red Hat suggest doing

  ```
  $ strings -f ⟨binary-name⟩ | grep hosts_access ←↩
  ```

  to see if a program is compiled with `libwrap`.
- Most programs are dynamically linked against `/usr/lib/libwrap.so.0`, so you can check for that with `$ ldd ⟨binary-name⟩ ←↩`
- Example:

  ```
  $ /usr/sbin/xinetd | grep libwrap ←↩
      libwrap.so.0 => /usr/lib/libwrap.so.0 (0x00320000)
  ```

- Older systems used `/usr/sbin/tcpd` and entered this in `/etc/inetd.conf` instead of the binary name of the service, but this is no longer necessary

Effect of SUID/SGID permissions

- A *program* with Set User-ID (SUID) permission will execute with the process owned by the owner of the file instead of the user that executed the program.
- A *program* with Set Group-ID (SGID) permission will execute with the group of the process the same as the group of the file instead of the group of the user that executed the program.
- A serious security risk.

Some History

- A friend in UNSW in 1985 used to stay in the lab with me till 5 AM many mornings; he had `root` access on the PDP-11, greatly upsetting the BOFH.
- He told me that he gained this through a set user-ID executable owned by `root`.

Specifying Permissions to `find`

- The `find` program finds files for which various conditions are true
- The `-perm` ⟨*mode*⟩ condition can find files which match the permissions specified in ⟨*mode*⟩ in various ways:
  - if ⟨*mode*⟩ starts with ‘−’ then true if *all* of the permissions in ⟨*mode*⟩ are present. Any permissions not in ⟨*mode*⟩ are ignored
  - if ⟨*mode*⟩ starts with ‘+’ then true if *any* of the permissions in ⟨*mode*⟩ are present. Any permissions not in ⟨*mode*⟩ are ignored
  - if ⟨*mode*⟩ starts with neither ‘−’ nor ‘+’ then true if permissions are exactly ⟨*mode*⟩.
- ⟨*mode*⟩ can be specified in octal or symbolically: e.g., you can specify `-perm +6000` or `-perm +ug=s`
  - both are true if the file has either SUID or SGID permission set.

`find`ing SUID or SGID files

- Here we can search the entire file system for SUID or SGID files: `$ find / -perm +6000 -ls` ↩

Why Verify Software Packages?

Main reasons:

- As another tool to check whether trojan executables have been installed by a cracker, replacing the original binary
- As a check that software downloaded from the Internet is from a trusted source and has not been compromised

Verify Package Files with `rpm`

- Ensure have the GPG key of the signer of the software packages, e.g.,

```
$ sudo rpm --import /media/cdrom/RPM-GPG-KEY* ↩
```

- Verify that each downloaded software package is signed before installing it: with `yum`, use the option `gpgcheck=1`
- If you have the RPM binary package file, you can check its signature with:

```
$ rpm -K ⟨complete-package-file-name⟩ ↩
```

- Example:

```
$ rpm -K httpd-2.0.54-10.2.i386.rpm ↩
httpd-2.0.54-10.2.i386.rpm:  (sha1) dsa
sha1 md5 gpg OK
```

Verify Installed Packages with `rpm`

- Do `$ rpm -V` ⟨*package-name*⟩ ↩
- Ensure that no binary executables have changed; here is an example of an executable that does not match the original installed version: `$ rpm -V spamassassin` ↩

```
S.5....T   /usr/bin/spamc
```

- This indicates that the size, the MD5sum and the timestamp have changed of this executable file, and it could quite possibly be a trojan
- There are eight characters; a dot indicates original value, a letter shows there is change:

S	file **S**ize differs
M	**M**ode differs (includes permissions and file type)
5	MD**5** sum differs
D	**D**evice major/minor number mismatch
L	read**L**ink(2) path mismatch
U	**U**ser ownership differs
G	**G**roup ownership differs
T	m**T**ime differs

Verify Packages with `apt/dpkg`

- To be done.
- There is a way…

Password Aging

- Limiting the age of passwords can improve security, although users may ping-pong between two passwords
- Best not to force users to change more than once every few months (page 607, [Gar2003]), else some will write them down

Password Aging options to `passwd`

- `-d` This is a quick way to disable a password for an account. It will set the named account passwordless. Available to root only.
- `-n` This will set the minimum password lifetime, in days, if the user's account supports password lifetimes. Available to root only.
- `-x` This will set the maximum password lifetime, in days, if the user's account supports password lifetimes. Available to root only.
- `-w` This will set the number of days in advance the user will begin receiving warnings that her password will expire, if the user's account supports password lifetimes. Available to root only.
- `-i` This will set the number of days which will pass before an expired password for this account will be taken to mean that the account is inactive and should be disabled, if the user's account supports password lifetimes. Available to root only.

Finding out about security alerts

- The best way to get cracked is to never apply security updates on a machine exposed to the Internet
- Subscribe to the mailing list for your distribution that announces security updates
- Subscribe to `http://lwn.net` and read their comprehensive security information, in particular from `http://lwn.net/security`

Update binaries with security alerts

- You can either apply updates automatically: with systems with `yum` installed, enable `yum` updates in `cron`.
- To update a system with `yum`:

```
$ sudo yum -y update ↩
```

- To update a system with `apt`:

```
$ sudo apt-get update ↩
$ sudo apt-get -y upgrade ↩
```

- If the system is mission critical and especially if it has complex software installed, install updates on a test system first

What are `iptables` and `ipchains`?

- Used to filter network packets coming into, out of and through the system
- Very useful for network security, Internet connection sharing
- `iptables` on 2.4, 2.6 kernels, `ipchains` on 2.2 kernels
- `iptables` is easier to use than `ipchains`
    - Many more things must be considered before you can predict what will happen to a packet passing through an `ipchains` system, while `iptables` tends to have a packet dealt with in one spot only, causing less brain pain.
- `iptables` has support for *stateful inspection* which allows the system to remember which response is in answer to which packet

Components of `iptables`

- There are four main terms to consider with `iptables`:

  table — a table holds a major category of set of rules.

  chain — sets of rules within a table that affect traffic

  rule — decides how to send a packet to a *target*. Next rule checks a packet if this doesn't match.

  target — can be `ACCEPT`, `DROP`, `QUEUE`, or `RETURN`. A matched packet is accepted, dropped, queued on another chain or returned to the parent chain from the current chain.

iptables tables

- There are three *tables* used by `iptables`:

  filter — default table for handling network packets

  nat — used to alter packets that create a new connection and used for Network Address Translation (NAT).

  mangle — for specific types of packet alteration, including time to live, type of service — for special routing purposes

iptables filter chains

- iptables filter table has three *chains*:

  INPUT for packets coming into the system, destined for the system itself

  OUTPUT for packets originating from the system, destined for outside the system

  FORWARD for packets entering the system that are meant for other systems on the other side, where the system is working as a router

Examples of filtering

- To drop all traffic to this machine from the source IP address 1.2.3.4, do:

```
$ sudo iptables -A INPUT -s 1.2.3.4 -j DROP ↩
```

- You might do that if there is nuisance traffic from that remote machine.

iptables nat chains

- The built-in chains for the nat table:

  PREROUTING — Alters network packets when they arrive

  OUTPUT — Alters locally-generated network packets before they are sent out

  POSTROUTING — Alters network packets before they are sent out

Viewing firewall rules

- To see the firewall rules for the `filter` table, do:

  ```
  $ iptables -L ↩
  ```

- To avoid the time to look up each IP address, do:

  ```
  $ iptables -L -n ↩
  ```

- To see the counters of the number of packets for each rule: `$ iptables -L -n -v ↩`
- To see the exact counters of the number of packets:

  ```
  $ iptables -L -n -v -x ↩
  ```

- To view the rules for the `nat` table without the DNS lookup delay:

  ```
  $ iptables -t nat -L -n ↩
  ```

- To view the rules for the `mangle` table without the DNS lookup delay:

  ```
  $ iptables -t mangle -L -n ↩
  ```

Sharing an Internet connection in an internal network

- Use masquerade where the external Internet address is changed by the ISP:

-
  ```
  iptables -t nat -A POSTROUTING
  -s 192.168.0.0/24 -o ppp0 -j MASQUERADE
  ```
- This is source Network Address Translation where the external address is changing.
- Where the Internet address is fixed, use the SNAT target:

-
  ```
  iptables -t nat -A POSTROUTING
  -s 192.168.0.0/24 -o ppp0 -j SNAT
  --to-source 220.233.65.75
  ```

Saving and restoring rules

- the `iptables-save` command saves the rules;
- `iptabels-restore` reads them back in from a file.
  - On Debian, need redirect to/from a file
  - Red Hat/Fedora systems store them in `/etc/sysconfig/firewall`

Perform security administration tasks I

Resources of interest

- *The `/proc` Filesystem*
  in `Documentation/filesystems/proc.txt` with Linux Kernel source
- Red Hat, Inc.
  Red Hat Enterprise Linux 4: Reference Guide
  Chapter 17: *TCP Wrappers and `xinetd`*
  `http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/ref-guide/ch-tcpwrappers.html`
  Chapter 18: *`iptables`*
  `http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/ref-guide/ch-iptables.html`
- Simson Garfinkel, Gene Spafford and Alan Schwartz.
  Practical Unix and Internet Security
  O'Reilly 2003.

Perform security administration tasks II

Resources of interest

- Olaf Kirch and Terry Dawson.
  Linux Network Administrator's Guide
  O'Reilly 2000.
  `http://tldp.org/LDP/nag2/`
- Info node: Find Permissions

  ```
  $ info '(find)Permissions' ↩
  $ info '(find)File Permissions' ↩
  ```
- rpm man page

  ```
  $ man rpm ↩
  ```
  and search for `VERIFY OPTIONS`
- Eric Foster-Johnson.
  RPM Guide
  `http://fedora.redhat.com/docs/drafts/rpm-guide-en/`

Topics Covered

Context

Objectives

Configuring TCP Wrappers

- Rules for `hosts.allow`, `hosts.deny`
- Format of rules in `hosts.{allow,deny}`
- Example
- Wildcards
- Patterns
- Option Fields: Logging
- `EXCEPT` operator
- How is TCP Wrappers enabled?

Finding files with SUID/SGID bit set

- Effect of SUID/SGID permissions
- Specifying Permissions to `find`
- Finding SUID or SGID files

Verify Packages

- Why Verify Software Packages?
- Verify Package Files with `rpm`
- Verify Installed Packages with `rpm`
- Verify Packages with `apt/dpkg`

Passwords and Aging Information

License Of This Document